

This lab session aims to explore the functionalities of the tool, **GNU Radio** which was introduced in Lab 2. **GNU Radio** is an open-source toolkit, through which various signal processing blocks of a communication system can be configured and developed in software. These blocks are typically constructed using hardware devices. In **GNU Radio**, a system can be made as a flowgraph containing blocks, each of which performs a specific task. Custom blocks can also be programmed using the **Python** programming language. It is assumed that all the necessary drivers such as **lib-io**, and **gr-io** etc. have been preinstalled following the steps in Lab manual 2.

1 Constructing a simple FSK radio system

Frequency-shift keying (FSK) is a type of frequency modulation technique in which the digital information is transmitted through a set of discrete frequencies around the centre frequency of the carrier wave. The simplest manifestation of FSK is the binary FSK (BFSK) where two frequencies are used to transmit the bits **0** (space frequency) and **1** (mark frequency).

In this section, we will simulate an FSK radio transmitter and receiver by following the steps originally laid out [here](#). This will help in familiarising the user interface of the development tool. The end result should appear something like what is shown in Fig. 1.

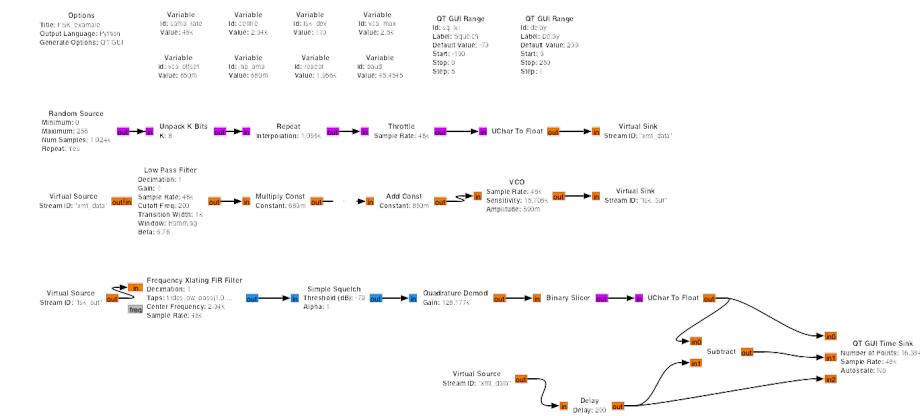


Figure 1: FSK Flowgraph in GNU Radio v3.8.

1.1 Creating Variables

Like in any development, the use of variables makes our job easy as the values can be easily reused any number of times. Moreover, if we want to simulate the system in different scenarios, we can easily do so when the variables are present.

Before we begin, it is important to know that there are different types of graphical user interfaces (GUIs) that exist in **GNU Radio**. We will use the **QT GUI** here. To do so, create a new file by going to

File → **New** → **QT GUI**. Once you obtain a new blank screen, look for the search icon (🔍) and click it. This will open up a search where different blocks can be quickly inserted.

First, we will insert the **Variable** block by dragging and dropping it from the list to the blank space. After double-clicking the variable block, we can configure the **id** and **value** of each variable.

- Create 8 new variables, the details of each are:

sample_rate	48000
centre	2210
fsk_dev	170
vco_max	2500
vco_offset	0.850
inp_amp	0.680
repeat	1056
baud	45.4545

The values above suggest a sampling frequency of 48 kHz, the central frequency of the carrier signal 2.21 kHz. The deviation around the chosen central frequency is 170 Hz which determines the frequencies of the mark and space components. These parameters are used in the US version of the Baudot Radioteletype (RTTY). RTTY is a telecommunication system that initially emerged in the late 1800s. Personal computers today are the contemporary manifestation of the RTTYS. The system works on a binary logic (mark and space) and on the transmitter side, FSK is used for modulating the audio tones. In the original (*Baudot*) version of RTTY, 8 bits are used out of which 5 are for data, 1 is for start and 2 are stop bits. The character size is enough to transmit alphabetical letters **A-Z** and digits **0-9**. A typical RTTY signal is shown in Fig. 2. A bit time of 22 ms is used for

amateur radio transmissions, which leads to a baud rate:

$$\text{baud} = \frac{1}{\text{bit time}} = \frac{1}{0.022} = 45.4545 \text{ Hz}$$

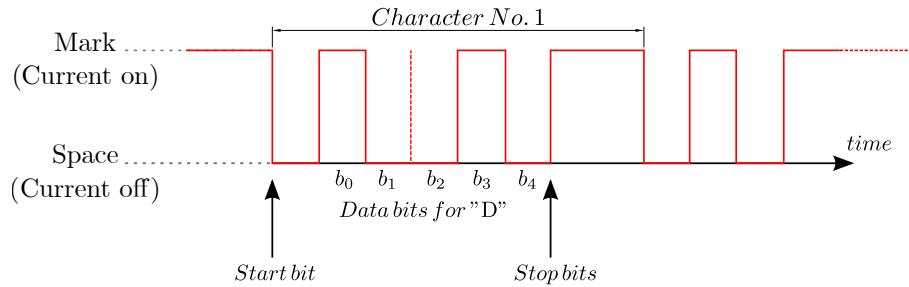


Figure 2: An RTTY data stream for the alphabet D.

The variable, `repeat` refers to the repeat factor which determines the total time of the data frame. In this example, we set it as an integer value:

$$\text{repeat factor} = (\text{int}) \text{ samp_rate} \times \text{bit time} = 48,000 \times 0.022 = 1056$$

In order to select the VCO parameters, we choose a full-scale frequency of 2.50 kHz. Although any value greater than the Mark frequency (in this case, 2.295 kHz) can be chosen, 2.50 kHz is a rounded number. The delta, Δ for the VCO is calculated as:

$$\Delta = \frac{\text{fsk_dev}}{f_{\text{full-scale}}} = \frac{170}{2500} = 0.068$$

The space frequency is created by the:

$$\text{vco_offset} = \frac{f_{\text{Mark}}}{f_{\text{full-scale}}} + 0 \times \Delta = 0.850$$

Similarly, the Space frequency is created by, The space frequency is created by the:

$$\text{inp_amp} = \frac{f_{\text{Mark}}}{f_{\text{full-scale}}} + 1 \times \Delta = 0.918$$

1.2 Simulating the Transmitter

The transmitter can be implemented by the following steps:

1. Create a `Random Source` that generates a byte with values ranging from `0 - 255`. Ensure that the output type is set to `byte`.
2. Create a `Unpack K bits` block that separates each bit of the input byte and creates a distinct byte out of it. Set the `K` value to `8`.
3. Since we are simulating a virtual source, we would like to limit the data flow. For this, the `Throttle` block is created.
4. Before we perform any mathematical operations such as filtering, we need to convert the data type of the input stream. We use a `UChar to Float` block.
5. In order to keep the flowgraph nice, easy to comprehend, we can create a component in multiple lines. We can do this by using the `Virtual Sink` and `Virtual Source` blocks. These blocks serve as connectors. We need to make sure the `Stream ID` matches for a given pair
6. For the virtual connectors created above, set the `Stream ID` to `''xmt_data''`.
7. Next, create a `Low Pass Filter` block which reduces the bandwidth the generated signal. The parameters to be entered are, `FIR Type - Float - Float (Decimating)`, `Decimation = 1`, `Gain = 1`, `Sample Rate = sample_rate`, `Cutoff Freq = 200`, `Transition Width = 1000`, `Window = Hamming` and `Beta = 6.76`.
8. The next involves calibrating the signal to the values so that amplitude is either at the `Mark` or `Space` frequency voltage levels. For this we use the `Const Multiply` and `Const Add` blocks for which the Const values are `inp_amp` and `vco_offset` respectively.
9. The last step is the most important in which the processed signal is passed to a `VCO` block. Here the two voltage levels are converted to the `Mark` and `Space` frequencies around the centre frequency.

The parameters to be inserted are, `Sampling Rate = sample_rate`, `Sensitivity = 15708`, and `Amplitude = 0.5`.

10. Create a `Virtual Sink` that is akin to labelling the transmitted signal. We will use it in the receiver module. Use the `Stream ID = 'fsk_out'`.

The transmitter module should resemble the second and third rows of the flowgraph shown in Fig. 1. Note that each block needs to be connected with the adjacent ones. In case there is an error, the arrow becomes red. Make sure that the data types of each block are the same. Also, note how we have implemented the FSK modulation through the calibration of the VCO.

1.3 Simulating the Receiver Module

A typical receiver module in a communication system consists of a filter, demodulation, and conversion components. Let's implement these step by step.

1. Create a `Virtual Source` with to capture the transmitted signal. Make sure to use the same `Stream ID` as defined in the transmitter side.
2. We now use a `Frequency Xlating FIR Filter` block that performs a frequency translation on the received signal. Through this, we can pick up a selected frequency band, translate it to the origin (in this case centre it along the centre frequency), and then remove the remaining frequency components. The parameter values used are, `Type = Float - Complex (Complex Taps)`, `Decimation = 1`, `Taps = firdes.low_pass(1.0,sample_rate,1000,400)`, `Center Freq = centre`, `Sample Rate = sample_rate`. Here we have utilised the feature to program a block using `Python` code.
3. Next, create a noise removal `Simple Squelch` block that sets a threshold level below which all the signal is removed. Set the `Threshold (dB) = 70`, and keep `Alpha = 1`.
4. The `Quadrature Demod` block is used to *demodulate* the received signal. It accepts a complex baseband signal which is then converted into

an output, that is the signal frequency in relation to the sample rate, multiplied with the gain. We obtain a signal which is positive for the input frequencies above zero and vice versa. Set the **Gain = 128177**.

- The **Binary Slicer** module binarises the signal into a byte of **1** and **0**.
 - Create a **UChar to Float** module to convert the byte into a numerical value for further processing.
 - Create a **Virtual Source** with the **Stream ID = 'xmt_data'**, which is followed by a **Delay** block having value **200**.
 - For the purpose of comparison, create a **Subtract** block with the inputs coming from the previous two steps.
 - To visualise the results, create a **QT GUI Time Sink** with three inputs, that come from the last three steps.

1.4 Results

In this section, we require you to paste the results of the simulation created in the sections above.

Paste the waveform of the transmit and received signals:

2 Exercise

1. Plot the frequency spectrum of:
 - (a) VCO output
 - (b) Xlating FIR Filter
 - (c) Digital received signal
2. What happens if you change the `Centre Frequency` parameter of the `Xlating FIR Filter` block to `0`?
 - (a) Plot the frequency spectrum in this case. Also, briefly explain what happens.
 - (b) Can you explain WHY do we need to keep the `Center Frequency = 2.21 kHz`?